23장 실행 컨텍스트

23장. 실행 컨텍스트(execution context)

실행 컨텍스트는 자바스크립트의 동작 원리를 담고 있는 핵심 개념

- 스코프를 기반으로 식별자와 식별자에 바인딩된 값을 관리하는 방식
- 호이스팅이 발생하는 이유
- 클로저의 동작 방식
- 테스크 큐와 함께 동작하는 이벤트 핸들러와 비동기 처리의 동작 방식

23.1 소스코드의 타입

자바스크립트는 4가지 소스코드 타입을 가지며, 각 실행 컨텍스트를 생성

1. 전역코드

- 전역에 존재하는 소스코드
- 전역에 정의된 함수, 클래스 등의 내부 코드는 포함되지 않는다.
- 전역 변수를 관리하기 위해 최상위 스코프인 전역 스코프를 생성
- var 키워드로 선언된 변수, 함수 선언문으로 정의된 전역함수
- 이들을 전역 객체의 프로퍼티와 메서드로 바인딩과 참조를 위해 연결되어야 한다.

이를 위해서 전역 코드가 평가되면 전역 실행 컨텍스트가 생성

2. 함수코드

- 함수 내부에 존재하는 소스코드
- 함수 내부에 중첩된 함수, 클래스 등의 내부 코드는 포함되지 않는다.
- 지역 스코프를 생성

- 지역 변수, 매개 변수, argument 객체를 관리
- 생성할 지역 스코프를 전역 스코프에서 시작하는 스코프 체인의 일원으로 연결해야 한다.

이를 위해 함수코드가 평가되면, 함수 실행 컨텍스트가 생성

3. eval 코드

- 빌트인 전역 함수인 eval 함수에 인수로 전달되어 실행되는 소스코드
- strict mode에서 자신만의 독자적인 스코프를 생성

이를 위해 eval 실행 컨텍스트가 생성

4. 모듈 코드

- 모듈 내부에 존재하는 소스코드
- 모듈 내부의 함수, 클래스 등의 내부 코드는 포함되지 않는다.
- 모듈로 독립적인 모듈 스코프를 생성한다.

모듈 실행 컨텍스트가 생성

23.2 소스코드의 평가와 실행

- 모든 소스코드는 실행에 앞서 평가과정을 거쳐 코드를 실행하기 위한 준비과정을 거침
- 소스코드는 소스코드 평가와 소스코드 실행 단계의 과정으로 나누어 처리

1. 소스코드 평가 과정

- 실행 컨텍스트 생성
- 변수, 함수 등의 선언문만 먼저 실행
- 생성된 변수, 함수 식별자를 실행 컨텍스트가 관리하는 스코프(렉시컬 환경의 환경 레코드)에 등록

2. 소스코드 실행

- 소스코드 평가가 끝난 다음에 순차적으로 실행됨(런타임이라고 함)

- 실행에 필요한 정보(변수나 함수의 참조를 실행컨텍스트가 관리하는 스코프에서) 검색해 취득
- 변수 값의 변경 등 소스코드의 실행 결과는 다시 실행 컨텍스트가 관리하는 스코프에 등록

23.3 실행 컨텍스트의 역할

코드가 실행되려면 스코프, 식별자, 코드 실행 순서 등의 관리가 필요하다.

1. 선언에 의해 생성된 모든 식별자를 스코프를 구분하여 등록하고 상태 변화를 지속적으로 관리
2. 스코프는 중첩관계에 의해 스코프 체인을 형성해야 함으로 스코프 체인을 통해 상위 스코프로 이동하며 식별자를 검색할 수 있어야 한다.
3. 현재 실행 중인 코드의 실행 순서를 변경(함수 호출에 의해 순서 변경)할 수 있어야 하며 다시 돌아갈 수 있게 해야한다.

실행 컨텍스트란?

23.4 실행 컨텍스트 스택

- 코드의 순서를 관리
  - 소스코드가 평가되면 실행 컨텍스트가 생성되고 실행 컨텍스트 스택의 최상위로 쌓인다.
  - 최상위 실행 컨텍스트는 언제나 현재 샐행 중인 컨텍스트이다.(실행 중인 실행 컨텍스트(running execution context))
- 자바스크립트 엔진은
  1. 먼저 전역 코드를 평가하여 전역 실행 컨텍스트를 생성한다.
  2. 함수가 호출되면 함수 코드를 평가하여 함수 실행 컨텍스트를 생성한다.
- 전역, 함수 실행 컨텍스트는 스택 자료구조로 관리 된다.

23.5 렉시컬 환경

- 식별자와 식별자에 바인딩된 값, 그리고 상위 스코프에 대한 참조를 기록하는 자료구조 실행 컨텍스트를 구성하는 컴포넌트
    - 스코프와 식별자를 관리
    - 스코프를 구분하여 식별자를 등록하고 관리하는 저장소 역활을 하는 렉시컬 스코프의 실체
    - Environment Record와 OuterLexicalEnvionmentReference 컴포넌트로 구성

Environment Record와 OuterLexicalEnvionmentReference 컴포넌트

- Environment Record(환경 레코드)
    - 스코프에 포함된 식별자를 등록하고 등록된 식별자에 바인딩 된 값을 관리하는 저장소
    - 환경 레코드는 소스코드의 타입에 따라 관리하는 내용에 차이가 있다.
- OuterLexicalEnvionmentReference(외부 렉시컬 환경에 대한 참조)
    - 상위 스코프를 가리킨다.
    - 해당 실행 컨텍스트를 생성한 소스코드를 포함하는 상위 코드의 렉시컬 환경을 말한다.
    - 이 것을 통해 단반향 링크드 리스트인 스코프 체인을 구현한다.

23.6 실행 컨텍스트의 생성과 식별자 검색 과정

```
var x= 1;
const y = 2;

function foo(a){
  var x = 3;
```

```
  const y =4;

  function bar(b){
    const z = 5;
    console.log(a + b + x + y +z);
  }
  bar(10);
}
foo(20);
```

위 코드의 실행 컨텍스트의 과정

1.전역 객체 생성

2.전역코드 평가

3. 전역 코드 실행

4.foo 함수 코드 평가

5. foo 함수 코드 실행

6. bar 함수 코드 평가

7. bar 함수 코드 실행

8. bar함수코드 실행 종료

9. foo함수 코드 실행 종료

10. 전역 코드 실행 종료

1. 전역 객체 생성

2. 전역 코드 평가

- 전역 코드 평가 자세한 과정
  1. 전역 실행 컨텍스트 실행
  1. 전역 렉시컬 환경 생성

1. 전역 실행 컨텍스트 실행

비어 있는 전역 실행 컨텍스트를 생성하여 전역 실행 컨텍스트를 생성한다.

(이때 실행중인 실행 컨텍스트가 된다.)

2. 전역 렉시컬 환경 생성

전역 렉시컬 환경을 생성하고 전역 실행 컨텍스트에 바인딩한다.

- **2-1. 전역 환경 레코드 생성**

  전역 환경 레코드 : 전역 변수를 관리하는 전역 스코프, 전역 객체의 빌트인 전역 프로퍼티와 빌트인 전역 함수, 표준 빌트인을 제공한다.

  이때, let,const키워드로 선언한 전역 변수는 전역 객체 프로퍼티가 되지 않고 개념적인 블록 내에 존재한다.

  이것을 구분하고 관리하기 위해 객체 환경 레코드(Object Environment Record)와 선언적 환경 레코드(Declarative Environment Record)로 구성된다.

  - 2-1-1. 객체 환경 레코드
    - var 키워드, 선언문으로 정의된 함수가 객체 환경 레코드에 연결된 BindingObject를 통해 전역 객체의 프로퍼티와 메서드가 된다.
    - 변수는 이때 var키워드 변수는 선언과 초기화 단계가 동시에 일어나면서 암묵적으로 'undefined'값을 바인딩한다.(변수 호이스팅이 일어나는 이유)
    - 함수는 BindingObject를 통해 객체를 즉시 할당받아 함수 호이스팅이 가능하다.
  - 2-1-2. 선언적 환경 레코드
    - let, const키워드로 선언한 전역 변수가 등록되고 관리된다.
    - const키워드로 선언한 변수는 선언단계와 초기화 단계가 분리되어 진행되어 런타임에 실행 흐름이 변수 선언문에 도달하기 전까지 일시적 사각지대에 빠진다.

- **2-2. this 바인딩**
  - 일반 적으로 전역 코드에서는 this는 전역 객체를 가르킨다.
  - [[GlobalThisValue]] 내부 슬롯에 this가 바인딩 된다.
    - 전역 환경 레코드와 함수 환경 레코드에만 존재

      (객체 환경 레코드와 선언적 환경 레코드에는 없다.)

- **2-3. 외부 렉시컬 환경에 대한 참조 결정**
  - 현재 평가 중인 소스코드를 포함하는 외부 소스코드의 렉시컬 환경, 즉 상위 스코프를 가리킨다.
  - 이를 통해 단방향 링크드 리스트인 스코프 체인을 구현한다.

- 전역 레시컬 환경은 외부 환경에 대한 참조에 null이 할당
- 전역 렉시컬 환경이 스코프 체인의 종점에서 존재함을 의미

3. 전역 코드 실행

- 전역 코드를 순차적으로 실행한다.
- 변수 할당문이 실행 되어 변수(x,y)에 값이 할당된다.
- foo함수가 실행된다.이때 식별자 결정을 하게 되며 전역 코드가 실행된다.
- 식별자 결정 : 어느 스코프의 식별자를 참조하면 되는지 결정하는 것

식별자 결정의 위해 식별자를 검색할 때는 실행 중인 컨텍스트에서 식별자를 검색하기 시작한다.

4. foo함수 코드 평가

- 함수 코드 평가 자세한 과정
  1. 함수 실행 컨텍스트 실행
  1. 함수 렉시컬 환경 생성

1. 함수 실행 컨텍스트 실행

- foo함수를 실행 컨텍스트를 생성한다.
- 이때 실행 컨텍스트 스택의 최상위, 즉 실행중인 컨텍스트가 된다.

2. 함수 렉시컬 환경 생성

foo함수 렉시컬 환경을 생성하고 foo함수 실행 컨텍스트에 바인딩한다.

환경 레코드와 외부 렉시컬 환경에 대한 참조로 구성된다.

- **2-1.함수 환경 레코드 생성**
  - 매개변수, argument객체, 함수 내부에서 선언한 지역 변수와 중첩 함수를 등록하고 관리한다.
- **2-2.this바인딩**
  - [[ThisValue]]내부 슬롯에 this가 바인딩된다.
  - this는 함수 호출 방식에 따라 결정된다.

- 여기에서는 foo함수가 일반함수호 호출되었기 때문에 this는 전역객체를 가르킨다.
- this를 호출하게 되면 바인딩된 객체가 반환된다.

- **2-3. 외부 렉시컬 환경에 대한 참조 결정**

  foo함수 정의가 평가된 시점에 실행 중인 실행 컨텍스트의 렉시컬 환경의 참조가 할당된다.

  - foo함수 정의는 전역 코드 평가 시점에서 평가된다.
    - 함수는 어디서 호출했는지가 아니라,

      어디에서 정의했는지에 따라 상위 스코프를 결정한다.
  - 상위 스코프는 내부 슬롯 [[Environment]]에 저장된다.
  - 내부 슬롯 [[Environment]]와 렉시컬 스코프는 클로저를 이해할 수 있는 중요한 단서이다.

5. foo함수 코드 실행

1. 런타임이 시작되어 foo함수의 소스코드가 순차적으로 실행된다.
2. 매개변수에 인수가 할당되고, 변수 할당문이 실행되어 지역변수 x, y에 값이 할당된다.
3. 함수 bar가 호출된다

- 식별자 결정을 위해 실행 중인 실행 컨텍스트의 렉시컬 환경에서 식별자를 검색하기 시작한다. 만약 찾는 식별자가 없다면 외부 렉시컬 환경에 대한 참조를 통해 식별자를 검색해 참조한다.

6. bar함수 코드 평가

- bar함수가 호출되면 bar함수 내부로 코드의 제어권이 이동한다.
- foo함수 코드 평가와 동일하게 진행된다.

7. bar함수 코드 실행

- 런타임 시작이 되고 bar 함수 코드가 순차적으로 실행된다.
- 매개변수에 인수가 할당되고, 변수 할당문이 실행되어 지역 변수 z값이 할당된다.
  - console와 log 식별자 검색

8. bar함수 코드 실행 종료

- console.log메서드가 호출되고 종료되면 더는 실행할 코드가 없어 종료하게 된다.
  - 이때 foo함수 실행 컨텍스트가 실행 중인 실행 컨텍스트가 된다.
  - bar 함수 코드를 다른 곳에서 참조한다면 bar함수 코드 실행 켄텍스트는 소멸하지 않는다.

9. foo함수 코드 실행 종료

- foo함수도 실행할 코드가 없어 종료한다.

10. 전역 코드 실행 종료

- foo함수가 종료되면 더는 실행할 전역 코드가 없으므로 전역 코드의 실행은 종료되고 전역 실행 컨텍스트도 실행 컨텍스트 스택에서 팝되어 실행 컨텍스트는 아무것도 남지 않는다.

23.7 실행 컨텍스트와 블록 레벨 스코프

- 선언적 환경 레코드를 갖는 렉시컬 환경을 새롭게 생성하여

  기존의 전역 렉시컬 환경을 교체한다.